# Petascaling and Performance Analysis of DALTON on Different Platforms

Simen Reine[(a)],Thomas Kjærgaard[(a)], Trygve Helgaker[(a)],
Olav Vahtras[(b,d)], Zilvinas Rinkevicius[(b,g)], Bogdan Frecus[(b)]
Thomas W. Keal[(c)], Andrew Sunderland[(c)], Paul Sherwood[(c)],
Michael Schliephake[(d)], Xavier Aguilar[(d)], Lilit Axner[(d)]
Maria Francesca Iozzi[(e)], Ole Widar Saastad[(e)],
Judit Gimenez[(f)]

[a] *Centre for Theoretical and Computational Chemistry (CTCC), Department of Chemistry, University of Oslo, P.O.Box 1033 Blindern, N-0315 Oslo, Norway*
[b] *KTH Royal Institute of Technology, School of Biotechnology, Division of Theoretical Chemistry & Biology, S-106 91 Stockholm, Sweden*
[c] *Computational Science & Engineering Department, STFC Daresbury Laboratory, Daresbury Science and Innovation Campus, Warrington, Cheshire, WA4 4AD, UK*
[d] *PDC Center for High Performance Computing at Royal Institute of Technology (KTH), Teknikringen 14, 100 44 Stockholm, Sweden*
[e] *University center for Information technology, University of Oslo, P.O.Box 1059 Blindern, N-0316 Oslo, Norway*
[f] *Computer Sciences - Performance Tools, Barcelona Supercomputing Center, Campus Nord UPC - C6, C/ Jordi Girona, 1-3, Barcelona, 08034*
[g] *KTH Royal Institute of Technology, Swedish e-Science Center (SeRC), S-100 44, Stockholm, Sweden*

**Abstract**

The work aims at evaluating the performance of DALTON on different platforms and implementing new strategies to enable the code for petascaling. The activities have been organized into four tasks within PRACE project: *(i)* Analysis of the current status of the DALTON quantum mechanics (QM) code and identification of bottlenecks, implementation of several performance improvements of DALTON QM and first attempt of hybrid parallelization; *(ii)* Implementation of MPI integral components into LSDALTON, improvements of optimization and scalability, interface of matrix operations to PBLAS and ScaLAPACK numerical library routines; *(iii)* Interfacing the DALTON and LSDALTON QM codes to the ChemShell quantum mechanics/molecular mechanics (QM/MM) package and benchmarking of QM/MM calculations using this approach; *(vi)* Analysis of the impact of DALTON QM system components with Dimemas. Part of the results reported here has been achieved through the collaboration with ScalaLife project.

## 1. Introduction

DALTON [1,2] is a versatile electronic structure program capable of performing quantum mechanics calculations using Hartree-Fock, Multiconfigurational Self Consistent Field and Coupled Cluster wave functions, and Density Functional Theory for molecular systems of various sizes.

DALTON program can perform the conventional tasks of quantum chemistry programs, like optimizing the geometry of a molecule or locating the transition state of a chemical reaction, but it's main strength is in the ability to compute spectroscopic constants of molecules and their optical, magnetic and mixed properties via a "fourth order response functions toolbox". The computational methods implemented in DALTON are suitable for the investigation of small to medium size molecular systems, consisting from a few to several hundreds of atoms. Hybrid quantum mechanics/molecular mechanics methods capable of handling complex systems, like proteins and organic crystals, are under development and expected to be included in future versions of DALTON program.

The results presented in this paper have been combined with the achievements reached within the ScalaLife [3] project where a snapshot of the latest version of DALTON quantum chemistry program has been extensively tested and analyzed within the ScalaLife project in order to reach the scalability objectives. A public release of the full DALTON code, named DALTON 2011, was provided in the summer 2011 and this code is available via the official program homepage [1,2] as well as through the ScalaLife project homepage [3]. Code modules and patches for the DALTON program developed within the ScalaLife project have been released shortly after the official release of DALTON 2011 via the ScalaLife homepage as they target only the latest version of DALTON code.

New features for this release developed within the ScalaLife project include:

- General handling of spin dependent perturbations in the RESPONSE module
- Quadratic response code for high spin open-shell systems
- New hyperfine coupling constants evaluation module into the RESPONSE module
- New I/O free exchange-correlation contributions integrator (AO basis)
- Multilayer parallelization of the SIRIUS module
- MPI code improvements in the RESPONSE modules
- Reduction of I/O bottlenecks for the LUSTRE file system

The DALTON 2011 suite also contains the new LSDALTON quantum chemistry code. The main strength of the LSDALTON program is the highly efficient and linear-scaling Hartree-Fock (HF) and Kohn-Sham (KS) Density functional theory (DFT) code suitable for large molecular systems, and the robust and efficient wave-function-optimization and response-optimization procedures.

We have also integrated the DALTON and LSDALTON QM codes into the ChemShell QM/MM package [4] for the first time. ChemShell is a computational chemistry environment which acts as a wrapper around external QM and MM codes to perform combined QM/MM calculations. Integration with ChemShell enables DALTON QM/MM calculations to be performed with a variety of MM approaches and provides supporting capability for automatic import of CHARMM and AMBER force fields, thus increasing the range of calculations that can be performed using DALTON on PRACE systems. In section 4 we detail the methods used to interface ChemShell to the DALTON codes and evaluate the performance of QM/MM calculations using this approach.

## 2. DALTON QM: Bottlenecks, performance improvements and first attempt of hybrid parallelization

### *2.1. Analysis of the code and identification of the bottlenecks*

The DALTON package has been constantly developed over several decades and during this period accumulated substantial amount of legacy code (subroutines, modules design, etc.), which is based on the acceptable programming practices at the development time, but considered inefficient or obsolete by the modern programming standards. Thus, the task of enhancing performance of the DALTON program on modern high performance systems requires not only optimization of computational algorithms for massive parallel execution, but also restructuring/refactoring of the DALTON code. The later task is a huge challenge in itself, as the DALTON program contains around 10 million of FORTRAN and C code lines. In this project we adapted an incremental code refactoring strategy and introduced changes only in code blocks, which directly contribute to HF wave function and DFT electron density optimization steps in the SIRIUS module, as these steps have been primary targets for more extensive parallelization and optimization in this project. During these efforts, the explicit declarations of all variables in the SIRIUS module and some parts of other relevant DALTON modules have been introduced and a new interface for MPI calls provided, which allows for creation and handling of MPI groups, instead of exclusive usage of MPI_COMM_WORLD group. The later modification allowed for introduction of layered MPI parallelization scheme in HF wave function and DFT density optimization algorithms, and aimed to resolve one of the major scalability bottlenecks of the DALTON code, namely the large communication latency between Master node and Slave nodes, when number of Slave nodes increases. This parallelization strategy and its performance during typical DFT calculations will be discussed in details in one of the following subsections.

Apart from effective parallelization strategy of selected algorithms it is also important to consider other possible bottlenecks of these algorithms, like disk access patterns. On high performance computer systems with network file systems, like LUSTRE, it's important to minimize number of disk access operations, as they become limiting factor defining overall performance of algorithms. To demonstrate this aspect of algorithm optimization, let us consider one of the major steps in DFT calculations, namely computation of exchange-correlation functional contribution to Kohn-Sham matrix. This task is typically accomplished by numerical integration exchange-correlation functional over predefined set of grid points. In conventional DALTON implementation, exchange-correlation functional integrator stores batches of grid points in binary direct access file associated with each MPI process and during integration loop small batches of grid points are read from file on each node. This design of algorithm is straightforward to implement and has also advantage of being algorithm with small memory footprint. On HPC systems with local disk attached to each node, the performance of above described algorithm with increasing number of MPI process degrades rather slowly and reasonable scalability can be achieved up to 500 CPU cores. Unfortunately, situation is entirely different on HPC systems with LUSTRE type file system, where severe degradation of performance of this integration code is already observed on relatively small number of CPU cores, like 16-120. To resolve this problem in ScalaLife project, we implemented new version of exchange-correlation functional integrator, which holds batches of grid points in memory and completely avoids disk access during integration loop. New integrator performs similarly to old

DALTON integrator on the systems with local disk attached to each node, while it outperforms old algorithm on the systems with LUSTRE file system. Furthermore, the scalability of integrator is overall improved and large number of CPU cores can be effectively exploited in performing this step of KS matrix formation. On this example we demonstrated importance of minimizing disk input/output for scalability and good performance of algorithms on modern HPC systems with network file systems. Following this initial success with exchange-correlation functional integrator we proceeded to eliminate I/O from Slave nodes in DALTON code. These modifications of code reduced I/O operations count on both Master and Slave nodes during execution of typical DFT calculations with DALTON program (see Table I) and thus made DALTON code more suitable for HPC systems with LUSTRE file system.

| I/O Access | DALTON 2011 (Released version) | | DALTON 2011 (Modified version) | |
|---|---|---|---|---|
| Node Type | Master Node | Slave Node | Master Node | Slave Node |
| Number of file open operations | 243 | 44 | 208 | 6 |
| Megabytes of data read/written during DFT calculation | 668.1 | 347.7 | 233.3 | 0.1 |

Table 1: Reduction of I/O access during typical DFT calculations with release and in this work modified version of DALTON program

### *2.2 Analysis of the software architecture*

The implementation of DALTON uses the task-parallelism of the computational model in order to distribute work amongst processors. This supports the implementation of the irregular structure of quantum mechanical computations that vary in time and size due to varying base function sets for different kinds of atoms. The implementation of a master-worker design pattern allows running the application with a very flexible number of nodes and achieves a good load-balance within the computational tasks.

Some disadvantage of this design become visible if several hundred processors are used in one simulation. For instance, the waiting time increases for workers in order to get response to their requests for new chunks of work or submitting results from the previous computational phase.

The performance analysis shows a decreasing parallel efficiency proportional to an increasing number of processors. This limits the reduction of computational time severely and underlines that it is of utmost importance to improve the scalability.

### *2.3 Improvement of the software architecture*

The optimizations in the application code are focused on the part for the calculation of the molecular properties, that is, the calculation of the response function. These function values consist of several contributions from different modeling approaches like quantum mechanics or density functional theory.

The bottleneck in the communication between master and worker processes has been removed through the introduction of a "team of masters". Before the optimization, all workers took part in the computation of every contribution to the response function although all these different contributions can be calculated independently and in parallel.

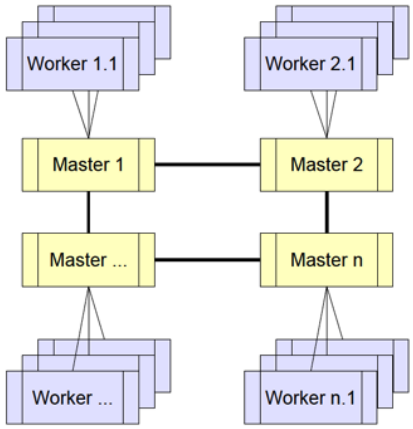


Figure 1: Master-worker relationship in a team of masters

Now after the refactoring, the available set of workers is partitioned into groups that have the task to compute only one contribution. Every group has a size adapted to the computational work needed for the calculation and its own master that collects the results. A substantial ease of the master's workload can be reached in that way. Smaller worker groups dealing with only one task lead to shorter waiting times for point to point communications as well as less complex and shorter collective communications. Furthermore, multiple workers can communicate at the same time achieving better use of interconnect and a higher degree of parallelism. The collected contributions are finally exchanged between the masters, which also compute the response function and initiate a new iteration until the convergence is reached.

### *2.4 Performance improvements*

The performance of the properties phase for both codes (the original and the optimized one) has been measured. The test case is the calculation of the g-tensor of di-tert-butyl nitroxide solvated in water that combines contributions from QM/MM and DFT calculations. Its strong scalability behaviour has been tested with runs from 128 to 1024 MPI-processes using the run with 128 ranks as base of the speedup calculations.

The optimized program version executes the QM/MM and DFT calculations in parallel. This includes also an optimized matrix composition routine that accelerates serial computations of the masters. The overall contribution to the performance improvement is 40% or even more depending on the number of used worker processes. The remaining improvement is gained from the parallel calculation of the different contributions to the response function.

The original program version scales until 512 cores (see Table 2) and provides so far a parallel efficiency around 50% (see Figures 2 and 3). Almost no speedup can be gained beyond this

number of nodes, whereas the optimized version runs now up to 1024 cores providing so far a parallel efficiency around 50% or more.

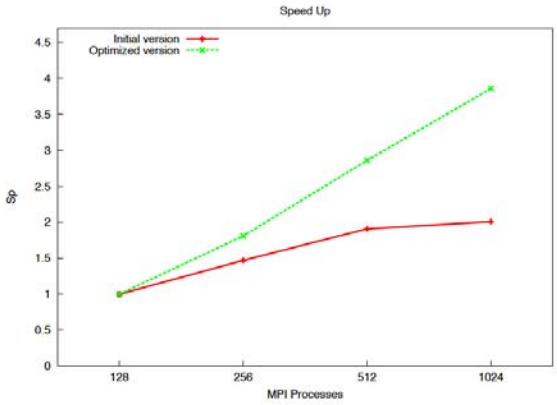


Figure 2: Speedup after optimization and redesign

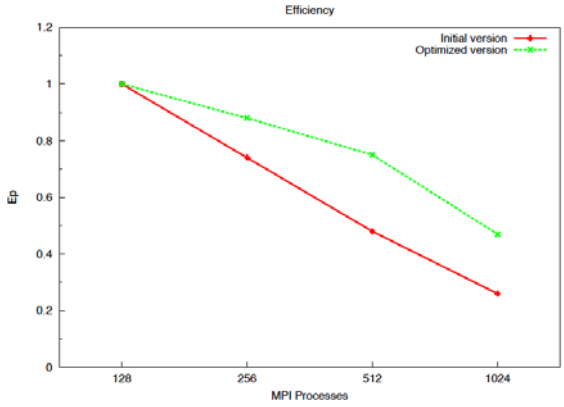


Figure 3: Parallel efficiency after optimization and redesign

| Num. Processes | Time | | Speedup | | Efficiency | |
|---|---|---|---|---|---|---|
| | Initial version | Optimized version | Initial version | Optimized version | Initial version | Optimized version |
| 128 | 24m 03s | 19m 19s | 1.00 | 1.00 | 1.00 | 1.00 |
| 256 | 16m 22s | 10m 40s | 1.47 | 1.81 | 0.74 | 0.88 |
| 512 | 12m 33s | 06m 45s | 1.91 | 2.86 | 0.48 | 0.75 |
| 1024 | 11m 30s | 05m 00s | 2.01 | 3.86 | 0.26 | 0.47 |

Table 2: Scalability of DALTON QM/MM before and after optimization and redesign

The percentages of computation time and time spent in MPI functions have also been improved by the new development. We found that our optimized version has increased its average computation time from the initial value of 70% for workers to 86%, reducing its average MPI time from around 30% to 14%. This implies that now the application does a better use of the resources and wastes less time in synchronization or communication.

## 3. LSDALTON: MPI integral components, optimization and scalability, implementation of Pure DFT

Quantum Chemistry applied to large molecular systems presents a unique set of challenges. The LSDALTON program is designed to treat such systems efficiently and in a linear-scaling fashion by employing state-of-the-art methodology [2]. Ongoing developments in computer technology have highlighted the need for highly scalable computational software, both with respect to memory usage and computational cost. It has been the goal of the current PRACE project to enable the LSDALTON code, comprising a little more than four hundred thousand lines of code, to take full advantage of contemporary supercomputer architectures. This is an ambitious undertaking, requiring algorithms at all levels of the code to be redesigned to exploit parallelism,

and in addition to greatly reduce memory requirements of the involved intermediates. The LSDALTON program was initially based on the DALTON program [1], and despite the fact that the LSDALTON program no longer has any code in common with the DALTON program, the collaboration remains strong, evident by the fact that the two codes are today distributed together. The main strength of the LSDALTON program is the highly efficient and linear-scaling HF and KS DFT code suitable for large molecular systems, and the robust and efficient wave-function-optimization and response-optimization procedures. The LSDALTON code (released as part of the DALTON program suite 2011) did not contain any MPI parallelization at the start of this project, and the goal of this project is to develop a fully MPI/OpenMP hybrid parallelization scheme that can fully utilize the PRACE HPC systems.

### *3.1. Parallelization approach*

The MPI parallelization approach adopted in the LSDALTON code can be categorized into two main categories, which naturally follows the underlying quantum-chemical code structure that basically boils down to the evaluation of various molecular integral components and a number of matrix operations. Efficient integral evaluation requires highly specialized routines and as a result specially tailored parallelization strategies, whereas the matrix operations are obvious candidates to parallelize using externally developed libraries.

The most time-consuming part of a DFT calculation is the integral evaluation - as needed for example for the KS matrix construction. The integral evaluation can be split into four contributions: the one-electron (h), Coulomb (J), exchange-correlation (xc) and exact exchange contributions. The Coulomb contribution can be evaluated exactly (regJ) or approximately using the much faster density-fitting technique (dfJ). The exchange-correlation contribution is evaluated using a numerical quadrature, whereas the remaining three terms, the one-electron, Coulomb and exchange contributions are evaluated using analytical expressions. In this project we only consider pure DFT for which the exact exchange contribution vanishes. The most challenging task of this sub-project is to efficiently MPI parallelize the analytical integral evaluation parts, whereas the numerical partitioning of the exchange-correlation contribution is expected to parallelize more easily.

For the analytical integral evaluation, we have developed our own MPI strategy where we divide the computations into tasks that rely on a priori predictions of the integral-evaluation cost. First, such partitioning requires only limited information passage between the nodes, and no information needs to be passed during the course of the actual integral evaluation. Secondly, it enables a fully distributed scheme, where only sub-blocks of involved matrices are needed on each node. Thirdly, for our integral-evaluation to be as efficient as possible large sets of integrals must be calculated simultaneously which is indeed also important for an efficient OpenMP evaluation of each individual task. Clearly our approach relies heavily on this partitioning scheme. The challenges with our scheme are twofold, namely to 1) give good time-estimates and 2) make a reasonable partitioning based on the time estimates. As such a significant amount of our time in this project has been devoted to ensure that both these two challenges have been addressed.

The numerical quadrature used in quantum chemistry is based on a superposition of atomic grid points, and for efficient evaluation batches of grid points are evaluated simultaneously. The grid

points are divided into homogeneous and predetermined subsets of grid points, such that each MPI node can evaluate a predetermined set of grid points, without the need for communication. The actual evaluation of the gridpoints on each MPI node is parallelized using OpenMP.

In addition to the integral-evaluation components, DFT relies heavily on linear algebra: for example for the energy optimization, molecular responses to external perturbations, geometry optimization routines and more. Although typically the matrix operations are not the time-limiting factor of the LSDALTON code (for serial calculations), the computational times are in many cases still significant, especially for large molecular systems (comprising tens of thousands of basis functions). A good parallelization approach for the matrix operations is therefore essential. Additionally, calculations on large molecular systems are limited on most system architectures by their associated memory requirements, which can easily reach 100 GB for calculations involving less than one thousand atoms. In order to perform large-scale calculations on a routine basis the memory requirements therefore need to be reduced. In this project we have interfaced our matrix operations to the PBLAS [5] and ScaLAPACK [6] numerical library routines, with assistance from STFC (the Science and Technology Facilities Council in the UK). This allowed both the efficient parallelization and a significant reduction of the overall memory requirements. The reduction in memory follows from the 2d block-cyclic matrix-distribution scheme employed by these libraries. As part of this project we have constructed the interface to PBLAS and ScaLAPACK in such a way that the various matrix operations in all parts of our code can switch between these and other library routines by simple input parameters.

We have also re-designed the integral code to enable a suitable memory-distributed format also for the analytical integral evaluation. This scheme relies on transformation between our memory distributed format and the ScaLAPACK format. These transformation steps have been implemented during the final phases of this project, but are not yet fully tested.

### *3.2. Performance and Results*

To determine the performance and scalability of the LSDALTON program we have chosen to look at the valinomycine (168 atoms) and insulin molecule (787 atoms, 7604 basis functions). The two molecules represent typical biochemical systems that we would like to investigate using the LSDALTON program, and for both molecules we have investigated the scalability of the different components of the code by performing typical Kohn—Sham energy optimization iterations using the pure DFT functional BLYP. All calculations have been conducted on the so-called ‘fat’ nodes on the CURIE supercomputer, where each node consists of 4 eight-core x86-64 CPUs. For the valinomycine molecule both pure MPI and hybrid MPI/OpenMP (with 8 threads) have been undertaken using two different basis sets (aug-cc-pVDZ and aug-cc-pVTZ, comprising 2604 and 5658 basis functions, respectively), and for insulin hybrid MPI/OpenMP calculations have been performed. The timings of the different components as well as overall speed-up and efficiency at different parallelization levels are summarized in Tables 3-8. The presented wall-clock timings are given in seconds. We note that for the hybrid MPI/OpenMP calculations, the presented number of cores (# cores) is the product of the number of MPI nodes and the number of OpenMP threads, and that the listed density-optimization step (Dens) is a conglomerate of different computational steps, all of which are dominated by matrix operations.

| *# cores* | *regJ* | *xc* | *Dens* | *Total* | *Speed-up* | *Ideal* | *Efficiency* |
|---|---|---|---|---|---|---|---|
| 4 | 40468 | 630 | 214 | 41312 | 1.0 | 1 | 100 |
| 32 | 5489 | 86 | 33 | 5608 | 7.4 | 8 | 92 |
| 64 | 2997 | 43 | 23 | 3063 | 13.5 | 16 | 84 |
| 128 | 2003 | 26 | 22 | 2051 | 20.1 | 32 | 63 |
| 256 | 1192 | 15 | 13 | 1220 | 33.9 | 64 | 53 |

Table 3: Pure MPI timings (in seconds) for valinomycine using BLYP/aug-cc-pVDZ with no density-fitting

| *# cores* | *dfJ* | *xc* | *Dens* | *Total* | *Speed-up* | *Ideal* | *Efficiency* |
|---|---|---|---|---|---|---|---|
| 4 | 206 | 630 | 214 | 1050 | 1.0 | 1 | 100 |
| 32 | 30 | 86 | 33 | 149 | 7.0 | 8 | 88 |
| 64 | 17 | 43 | 23 | 83 | 12.7 | 16 | 79 |
| 128 | 12 | 26 | 22 | 60 | 17.5 | 32 | 55 |
| 256 | 8 | 15 | 13 | 36 | 29.2 | 64 | 46 |

Table 4: Pure MPI timings (in seconds) for valinomycine using BLYP/aug-cc-pVDZ with density-fitting approximation (df-def2)

| *# cores* | *regJ* | *xc* | *Dens* | *Total* | *Speed-up* | *Ideal* | *Efficiency* |
|---|---|---|---|---|---|---|---|
| 32 | 9255 | 87 | 41 | 9383 | 1.0 | 1 | 100 |
| 256 | 1370 | 15 | 15 | 1400 | 6.7 | 8 | 84 |
| 512 | 817 | 9 | 14 | 840 | 11.2 | 16 | 70 |
| 1024 | 514 | 5 | 21 | 540 | 17.4 | 32 | 54 |
| 2048 | 340 | 5 | 10 | 355 | 26.4 | 64 | 41 |

Table 5: Hybrid MPI/OpenMP timings (in seconds) for valinomycine using BLYP/aug-cc-pVDZ with no density-fitting

| *# cores* | *dfJ* | *xc* | *Dens* | *Total* | *Speed-up* | *Ideal* | *Efficiency* |
|---|---|---|---|---|---|---|---|
| 32 | 57 | 87 | 41 | 185 | 1.0 | 1 | 100 |
| 256 | 10 | 15 | 15 | 40 | 4.6 | 8 | 58 |
| 512 | 7 | 9 | 14 | 30 | 6.2 | 16 | 39 |
| 1024 | 5 | 5 | 21 | 31 | 6.0 | 32 | 19 |
| 2048 | 4 | 5 | 10 | 19 | 9.7 | 64 | 15 |

Table 6: Hybrid MPI/OpenMP timings (in seconds) for valinomycine using BLYP/aug-cc-pVDZ with density-fitting approximation (df-def2)

| *# cores* | *dfJ* | *xc* | *Dens* | *Total* | *Speed-up* | *Ideal* | *Efficiency* |
|---|---|---|---|---|---|---|---|
| 32 | 183 | 268 | 645 | 1096 | 1.0 | 1 | 100 |
| 256 | 28 | 45 | 128 | 201 | 5.5 | 8 | 68 |
| 512 | 20 | 28 | 92 | 140 | 7.8 | 16 | 49 |
| 1024 | 15 | 19 | 64 | 98 | 11.2 | 32 | 35 |

Table 7: Hybrid MPI/OpenMP timings in seconds for valinomycine using BLYP/aug-cc-pVTZ with density-fitting approximation (cc-pVTZdenfit)

| *# cores* | *regJ* | *xc* | *Dens* | *Total* | *Speed-up* | *Ideal* | *Efficiency* |
|---|---|---|---|---|---|---|---|
| 256 | 3175 | 32 | 93 | 3300 | 1.0 | 1 | 100 |
| 512 | 1682 | 20 | 68 | 1770 | 1.9 | 2 | 93 |
| 1024 | 1004 | 15 | 49 | 1068 | 3.1 | 4 | 77 |
| 2048 | 599 | 12 | 131 | 742 | 4.4 | 8 | 56 |

Table 8: Hybrid MPI/OpenMP timings in seconds for insulin using BLYP/cc-pVDZ with no density-fitting

Good scalability is observed for the calculations using the exact integral evaluation, which can be seen from the listed efficiency of 41% at 2048 cores (relative to the 32 cores reference value) for valinomycine BLYP/aug-cc-pVDZ calculation, in Table 4, and the 56% efficiency at 2048 cores (relative to the 256 cores reference value) for the insulin BLYP/aug-cc-pVDZ calculation, in Table 8. By comparing tables 3 and 5 we see an overall speed-up of about a factor of four when going from a pure MPI to the 8-threaded hybrid MPI/OpenMP calculation (ranging from a factor 4.4 using 4 MPI nodes to a factor 3.4 using 256 MPI nodes).

The scalability for the calculations using the density-fitting approximation is reduced compared to the calculations based on the exact integral evaluation, with an efficiency of 19% at 2048 cores for the valinomycine BLYP/aug-cc-pVDZ(df-def2) calculation and 35% efficiency at 1024 cores for BLYP/aug-cc-pVTZ(cc-pVTZdenfit). This behaviour is expected since these approximated calculations are much faster than the exact calculations, and will naturally be dominated by computational overheads. If we isolate the integral components (by ignoring the density-optimization step) the efficiencies are instead 50% and 41%, respectively, which are both good. This indicates that we are limited by the scalability of the matrix operations, which are expected to greatly improve when going to larger systems. This can also be seen from the scalability of the various contributions illustrated in Figures 4-7. These figures depict the relative speed-up of each component for the four systems studied in Tables 3-8. We note that these results show that the numerical integral evaluation step (xc) scales well up to 2048 cores.

We note here that for the valinomycine BLYP/aug-cc-pVDZ (df-def2) 2048 core calculation, an internal investigation of our developed task partitioner has shown that the actual integral evaluation efficiency is about 80% - 90%, clearly indicating that this step is dominated by computational overheads. Indeed, this step is dominated by the transformation steps from our full integral-specific format to the PBLAS/ScaLapack distributed matrix format, and we expect this overhead to vanish upon completion of the integral-distributed to PBLAS/ScaLapack distributed matrix format.

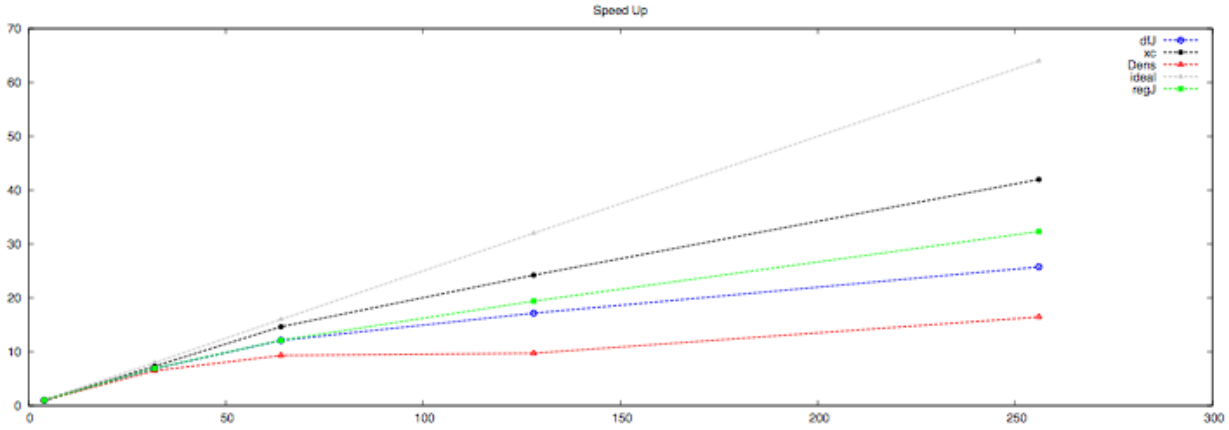


Figure 4: Pure MPI speed-up relative to the MPI/OMP 4/1 reference for valinomycine using BLYP/aug-cc-pVDZ (df-def2)

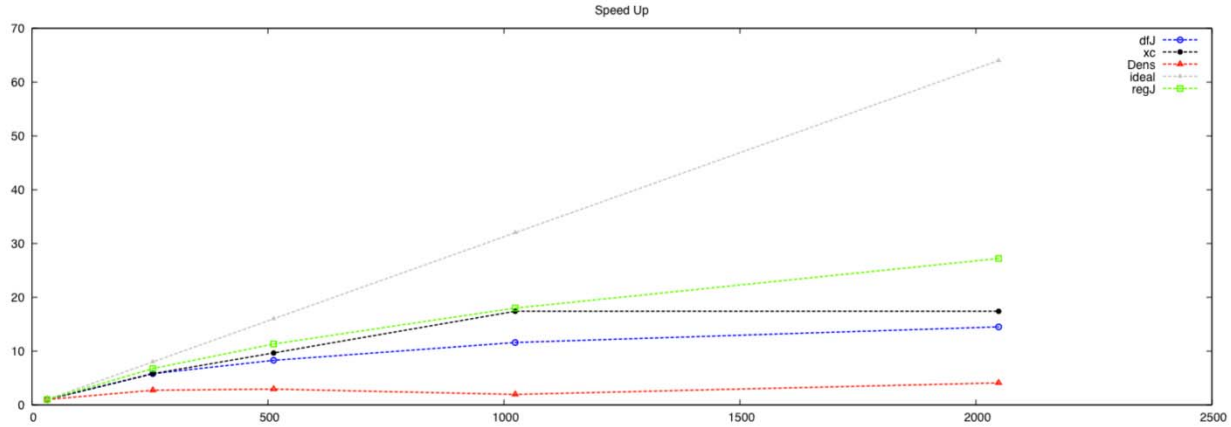


Figure 5: Hybrid MPI/OpenMP speed-up relative to the 4/8 reference for valinomycine using BLYP/aug-cc-pVDZ (df-def2)

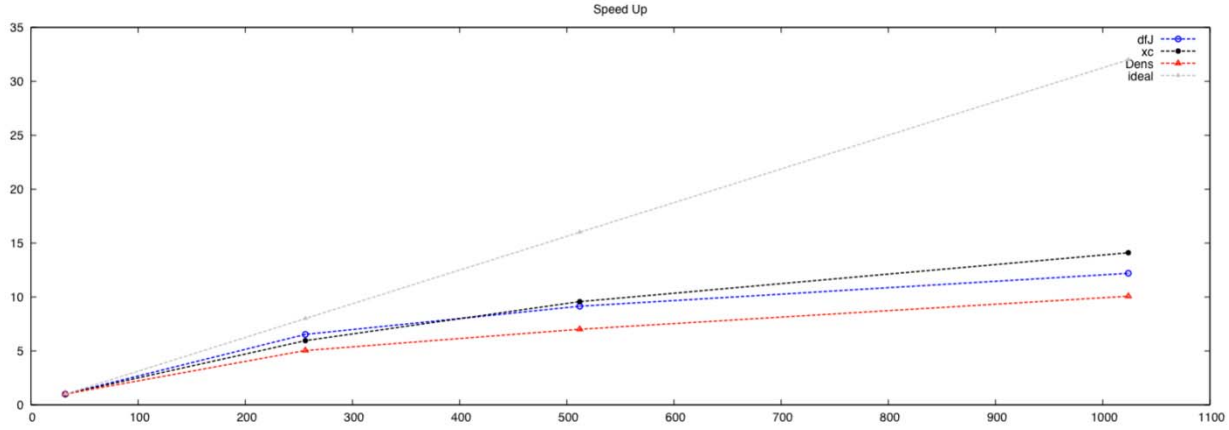


Figure 6: Hybrid MPI/OpenMP speed-up relative to the 4/8 reference for valinomycine using BLYP/aug-cc-pVTZ (cc-pVTZdenfit)

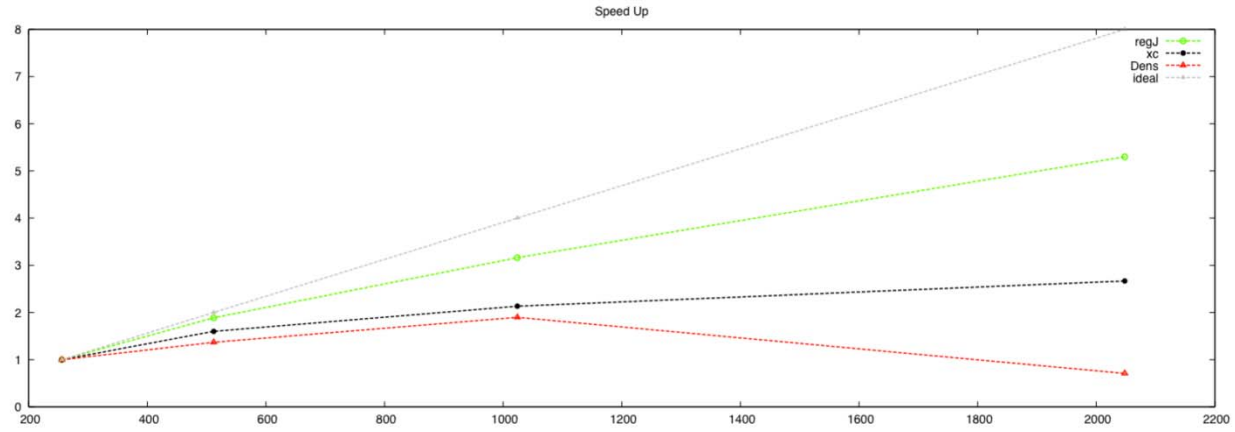


Figure 7: Hybrid MPI/OpenMP speed-up relative to the 32/8 reference for insulin using BLYP/cc-pVDZ

Overall, we are very satisfied with the developments conducted in this project. We have MPI parallelized three out of four integral components needed for hybrid DFT calculations, and have shown scalability up to 2048 cores. As a part of this project we have established the main code framework and strategies to exploit parallelism at all levels of the code. Thanks to the PRACE infrastructure we have attended a MPI profiling workshop, and we have been given access to supercomputing facilities where we could test and optimize the LSDALTON code. This has allowed us to identify and remove a series of computational overheads, and greatly enhance our understanding of what developments are needed to reach the overall goal of peta-scaling capabilities. We will continue to improve MPI performance to achieve this goal.

## 4. Interfacing DALTON and LSDALTON with the ChemShell QM/MM package

There are two possible ways to interface ChemShell with DALTON: a 'binary interface' and a 'linked-in interface'. Both methods involve the creation of a DALTON input file and parsing of the output file by ChemShell using scripts written in the Tcl programming language. The difference lies in the way DALTON is called. In the binary interface, an ordinary DALTON build is called using a shell command. This is straightforward to set up but it is not optimal for parallel execution as ChemShell and DALTON cannot share a single MPI environment. The 'linked-in interface' is a more sophisticated solution in which DALTON is compiled as a library that can be called directly from ChemShell. The MPI environment is then shared and advanced parallelization strategies such as task-farming [7] can then be employed. However, this approach requires modifications to the DALTON code, as detailed below.

In this project we have implemented both binary and linked-in interfaces to LSDALTON and DALTON.

### *4.1. LSDALTON interface*

A binary interface between ChemShell and LSDALTON has been written with support for energy and gradient calculations with and without a point charge environment. The LSDALTON

input file is generated by the ChemShell Tcl interface and the output is parsed to obtain the energy and gradient, which is then combined with output from the MM code to form the total QM/MM energy and gradient. This can then be passed to a high-level routine such as the DL-FIND geometry optimizer [8].

The DALTON 2011 release of LSDALTON does not support point charges in the QM Hamiltonian and so electrostatic embedding is not possible with this version. However, this feature has since been added to the development version of LSDALTON and so full QM/MM calculations with electrostatic embedding will be available to users in the next release. The interface is fully functional with the development version.

We have also developed a linked-in interface between ChemShell and the development version of LSDALTON. In this interface the work of the "lsdalton" run script in setting up the LSDALTON working environment is performed instead by the ChemShell Tcl interface and then LSDALTON is called directly as a subroutine from ChemShell. No modifications of LSDALTON were required to link in this way as LSDALTON is already set up internally as a collection of library files and a wrapper (ChemShell simply acts as the wrapper instead). ChemShell must be configured with a -with-lsdalton option to link in the LSDALTON libraries. A number of minor bugs were revealed in the DALTON 2011 version of LSDALTON which prevented multiple calls to the code from ChemShell (e.g. during a geometry optimization), but these have been fixed in the development version.

The main advantage of the linked-in interface is the ability to share the same MPI environment in ChemShell and LSDALTON. This required modifications to the LSDALTON code specifically for use with ChemShell, which are turned on using the –DVAR_CHEMSHELL flag during compilation. In essence, the MPI environment is controlled by ChemShell and therefore the MPI initialization and finalization steps in LSDALTON must be switched off. Instead, an MPI communicator is passed to LSDALTON, which is stored as MPI_COMM_LSDALTON and used by all MPI calls in the code (for convenience a FORTRAN module is used to store the communicator). This means that ChemShell can split MPI_COMM_WORLD and give LSDALTON a communicator representing a subset of the MPI environment, as in a task-farmed calculation [7]. It is essential that LSDALTON does not reference MPI_COMM_WORLD explicitly in MPI calls, but for a standard (non-ChemShell) build MPI_COMM_LSDALTON can be set equal to MPI_COMM_WORLD at initialization.

A small modification is also required to the new ScaLAPACK routines in LSDALTON for use with ChemShell. In the default ScaLAPACK initialization call (SL_INIT) a BLACS context equivalent to MPI_COMM_WORLD is returned, which is not compatible with a split communicator environment. However, in MPIBLACS, contexts are in fact equal to MPI communicators and therefore we can simply set the BLACS context to the communicator passed in from ChemShell. It should be noted that in non-MPI implementations of BLACS this would not be possible but we would then be unable to use ChemShell's task-farming methods anyway, as they are dependent on MPI, so this approach covers all use cases. The process grid is then initialized by a call to BLACS_GRIDINT. We have previously used this solution in STFC's GAMESS-UK QM [9] package and it has proved to be robust across multiple platforms.

### 4.2. DALTON interface

The binary interface between ChemShell and DALTON is fully functional, supporting energy and gradient calculations, restarts, and calculations with point charges. Although the interface is complete, QM/MM calculations using DFT are not possible with the DALTON 2011 release of DALTON, due to a bug in the handling of point charges. This has now been patched in the development version of DALTON and therefore QM/MM calculations with ChemShell will be fully supported in the next public release of DALTON.

We have also developed a linked-in interface between ChemShell and the development version of DALTON. As in the case of LSDALTON, the work of the "dalton" run script in setting up the DALTON working environment is performed by the Tcl interface and then DALTON is called directly as a subroutine from ChemShell. A small modification was required to compile DALTON as a subroutine rather than a standalone program, which is enabled using the –DVAR_CHEMSHELL flag during compilation. ChemShell must be configured with the -with-dalton option to link in the DALTON libraries.

The version of DALTON we had access during development does not support communicators other than MPI_COMM_WORLD, and so it is not currently possible to run task-farmed calculations with ChemShell/DALTON. However, the default MPI environment can still be shared between ChemShell and DALTON which is beneficial for ordinary parallel runs. Again, the MPI initialization and finalization steps are turned off using the –DVAR_CHEMSHELL flag. In future work we will be able to make use of the new task management developments in DALTON to share MPI communicators between ChemShell and DALTON and so to enable task-farming in the same way as the LSDALTON interface.

### 4.3. ChemShell benchmarks

To investigate the parallel scaling performance of the ChemShell interface we performed benchmark calculations on CURIE using LSDALTON. An MPI and ScaLAPACK -enabled build of LSDALTON was compiled on CURIE as described above and linked in to a parallel MPI build of ChemShell.

The benchmark system was a QM/MM model of the enzyme p-hydroxybenzoate hydroxylase (PHBH). The model consists of the protein, the oxidized form of the co-factor flavin adenine dinucleotide, substrate p-hydroxybenzoate and a surrounding shell of solvent molecules, giving a total of 22,748 atoms. Three QM regions were considered in order to test how the MPI scaling changes with system size. Region A consisted of the substrate, part of the co-factor and a neighboring amino acid residue (no. 293). Region B consisted of region A plus the rest of the co-factor. Region C consisted of region B plus several more amino acid residues (nos. 294-298).

As the exact exchange routines had not yet been optimized in the latest LSDALTON build, the benchmark calculations were performed the pure DFT functional BLYP. In all cases the TZVP basis set ("Turbomole-TZVP") and the df-def2 auxiliary basis for density fitting was used. Table 9 shows the resulting calculation sizes.

| QM region | QM atoms | Basis functions | Auxiliary basis functions | Quadrature grid points |
|---|---|---|---|---|
| A | 70 | 940 | 3568 | 878692 |
| B | 123 | 1706 | 6602 | 1560430 |
| C | 183 | 2443 | 9359 | 2289696 |

Table 9: Calculation sizes with the BLYP GGA functional, TZVP basis set and df-def2 auxiliary basis

The calculation timing results are presented in Table 10 as wall clock figures in seconds for a single point QM/MM energy and gradient calculation. The benchmarks were run on the CURIE 'large' nodes which consist of 4 eight core 2.27 GHz Nehalem-EX CPUs, giving 32 cores per node. In all the calculations we used the recommended configuration of one MPI process per node, each with 32 OpenMP threads. The ChemShell timing results are divided into an initialization stage (mainly force field setup) which is not parallelized, an LSDALTON step for the QM energy and gradient, and a DL_POLY step for the MM energy and gradient. We also consider separately the SCF and gradient calculation timings within LSDALTON.

| MPI Proc. | Total Cores | Chemsh. Init. / s | LSDALTON (QM) / s | | | dl_poly (MM) / s | Chemsh. Total / s | S(N/2) Chemsh | S(N/2) LSDAL. |
|---|---|---|---|---|---|---|---|---|---|
| | | | Total | SCF | Gradient | | | | |
| *QM Region A* | | | | | | | | | |
| 1 | 32 | 42 | 19102 | 12366 | 5554 | 9.8 | 19154 | - | - |
| 2 | 64 | 43 | 10141 | 6188 | 3227 | 5.3 | 10189 | 1.88 | 1.88 |
| 4 | 128 | 43 | 5686 | 3410 | 1823 | 3.0 | 5731 | 1.78 | 1.78 |
| 8 | 256 | 99 | 3465 | 2120 | 1004 | 1.9 | 3565 | 1.61 | 1.64 |
| 16 | 512 | 45 | 2250 | 1487 | 512 | 1.3 | 2296 | 1.55 | 1.54 |
| 32 | 1024 | 46 | 1440 | 1016 | 274 | 0.7 | 1487 | 1.54 | 1.56 |
| 64 | 2048 | 47 | 1174 | 812 | 157 | 0.4 | 1221 | 1.22 | 1.23 |
| *QM Region B* | | | | | | | | | |
| 4 | 128 | 42 | 50546 | 46416 | 3034 | 3.2 | 50591 | - | - |
| 8 | 256 | 44 | 30661 | 28217 | 1695 | 2.0 | 30706 | 1.65 | 1.65 |
| 16 | 512 | 44 | 15741 | 14344 | 915 | 1.0 | 15786 | 1.95 | 1.95 |
| 32 | 1024 | 45 | 11210 | 10029 | 536 | 0.6 | 11255 | 1.40 | 1.40 |
| 64 | 2048 | 46 | 9020 | 8354 | 293 | 0.3 | 9066 | 1.24 | 1.24 |
| *QM Region C* | | | | | | | | | |
| 16 | 512 | 42 | 53913 | 50754 | 1968 | 1.0 | 53956 | - | - |
| 32 | 1024 | 43 | 42538 | 40225 | 1267 | 0.5 | 42581 | 1.27 | 1.27 |
| 64 | 2048 | 44 | 28366 | 26796 | 747 | 0.3 | 28411 | 1.50 | 1.50 |

Table 10: Wall clock figures in seconds for a single point QM/MM energy and gradient calculations with ChemShell/LSDALTON

It is clear from the results that for all three QM regions, the QM step is by far the dominant part of the calculation. The ChemShell initialization step takes a constant 40-50 seconds (except for one anomalous result). Although not parallelized, the initialization overhead has a negligible effect on the total time to solution. The MM calculation using ChemShell's DL_POLY module is

parallelized (by MPI) and scales well up to 64 processes, but has a still smaller effect on the total time as even the serial calculation for region A takes no longer than 10 seconds. The scaling behavior of LSDALTON is therefore of most interest.

For a given QM region all LSDALTON calculations required the same number of SCF steps to converge (72 for region A, 148 for region B and 160 for region C), and therefore the timings can be compared directly. It was not possible to run baseline single MPI process benchmarks for all three regions due to the excessive wall time required, and therefore we compare parallel efficiency indirectly using "speedup compared to running with half the number of MPI processes" which we denote "S(N/2)" in the table. If ChemShell and LSDALTON were ideally parallelized S (N/2) would be 2 for every doubling of the number of MPI processes.

As can be seen from the table, the S (N/2) measure is quite volatile, but does give some indication of the relative parallel scaling behavior for each QM region. Because the LSDALTON calculation is the dominant factor in the scaling results, the S (N/2) results for LSDALTON are almost identical to S (N/2) for ChemShell as a whole. In Figure 8 we plot the LSDALTON S (N/2) scaling for all three QM regions:

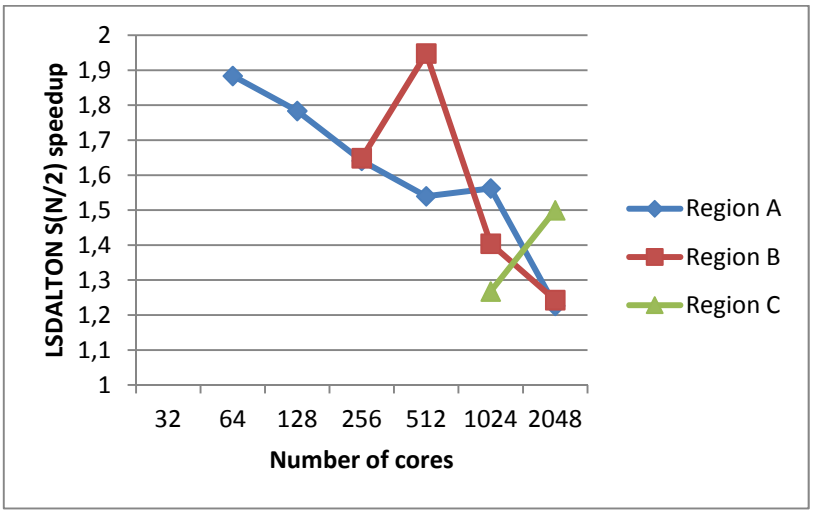


Figure 8: LSDALTON S(N/2) parallel scaling for QM regions A, B and C. Ideal scaling is equivalent to S(N/2)=2.

With the exception of the remarkably good result for the 512 core Region B calculation, the results indicate that the MPI scaling for region A and B calculations are similar, and both scale reasonably well up to about 1024 cores. For region C the trend is not clear as results were not available below 512 cores and the 1024 core result appears surprisingly slow, but they suggest that good scaling may be possible up to 2048 cores or more. Further benchmarks would be required to confirm whether larger QM regions lead to improved scaling in general.

In the Table 10 the LSDALTON timings for the SCF iterations and gradient evaluation have also been quoted to check that each stage of the calculation scales well. The results show that both SCF and gradients have similar scaling properties to the overall LSDALTON calculation.

It should be stressed that LSDALTON has yet to be optimized for QM/MM calculations (in particular the evaluation of the core Hamiltonian contribution) and so further improvements to the MPI scaling can be made in future. Another way to improve parallel scaling with ChemShell

is to add a second layer of parallelism using task-farming [7]. This is not applicable to single point calculations such as the benchmarks above, but for other common chemical tasks such as Hessian evaluations and nudged elastic band optimization task-farming offers a straightforward way to scale to much higher core counts.

## 5. Analysis of the impact of DALTON QM system components with Dimemas

The objective of this analysis was to complement the performance analysis carried out within the scope of ScalaLife project. DALTON code is composed by two different phases and the original plan was to study both of them. The study has been concentrated on the first phase that computes the wave function as the properties that were improved within the framework of ScalaLife were not yet available on the DALTON distribution and we considered it was meaningless to analyze the older version with well-known limitations already optimized.

The trace file we have worked with corresponds to an execution of DALTON with 256 tasks with a relevant input case. From this trace file we isolated one of the iterations of the wave function phase as the target of our parametric study. It may be interesting to extend the analysis to different tasks sizes to identify how the resource requirements evolve when increasing the scale.

The goal of the parametric study has been to evaluate the sensitivity to different architectural parameters. We have focused on 3 parameters that we considered may be the most relevant. The first two parameters, network latency and bandwidth allow us to evaluate the network requirements of the code, what would be the penalty if we move to a machine with a low network and to measure what would be the potential benefit from moving to a network with a larger bandwidth. The third parameter is that we name the CPU ratio that corresponds to the speed-up of the processor, it allow us to evaluate the impact of moving to a faster processor or parallelizing the work done by the MPI tasks (keeping the same communication requirements).

The results are shown on the next tree figures that correspond to the variations of 2 parameters keeping the third one constant. The Figure 9 corresponds to the scenario with a fixed latency of 4 microseconds that is quite similar to the current latency on the machine used to obtain the trace file. We can see how both the CPU ratio and the bandwidth have an impact on the achieved speed-up. With networks slower than 512 MB/s there would be no benefits on the code even if the processor is 64 times faster than the current machine. On the other side, to see significant benefits on the bandwidth at least an 8 times faster CPU is required. This last observation indicates that it would be important to optimize or parallelize the current sequential computations to benefit from future faster networks.

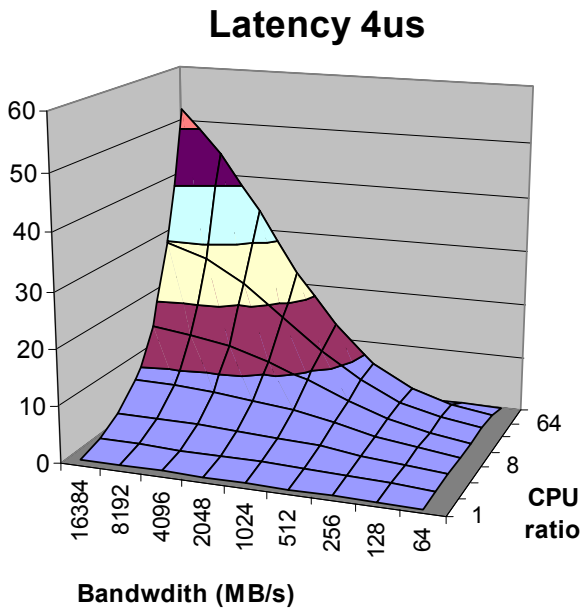


Figure 9: Fixed latency of 4 microseconds at the vvariations of 2 parameters keeping the third one constant

The Figure 10 corresponds to the scenario with an improved processor that is 8 times faster than the one used for obtaining the trace file. This scenario focuses on the impact of improving the computing time with no changes on the network requirements. The first observation is that while with the previous scenario we were able to obtain a speed-up up to 50 times faster than the current execution, this plot only goes up to 8. Again this is indicating a problem with the compute regions. The fact that all the combinations have a speed-up over the reference execution is because of the 8 times faster processor. We can see that in this case there is almost no impact from variations of the latency. Only when the bandwidth is larger than 4GB/s we can start to see a very small benefit from reducing the latency. We can see the benefits of increasing the network bandwidth on this scenario and it seems that with 16GB/s is starting to reach the plateau so the application would not be benefited from faster networks.

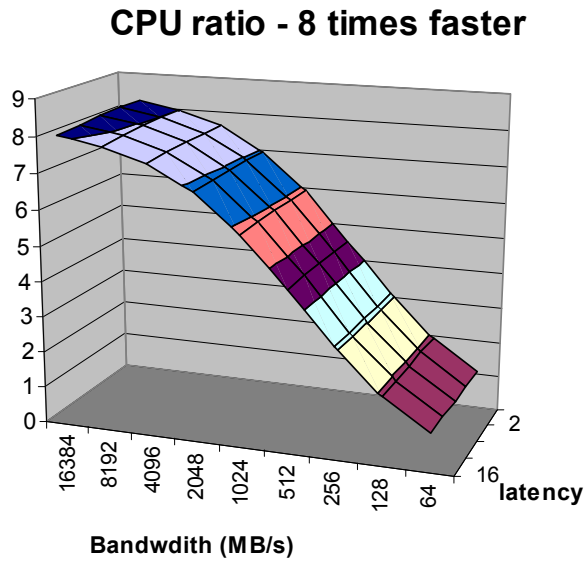


Figure 10: The impact of improving the computing time with no changes on the network requirements

The Figure 11corresponds to the scenario with a fixed bandwidth of 1GB/s that is quite similar to the current bandwidth on the machine used to obtain the trace file. Again we can see almost no effect to latency variations with a fixed bandwidth except for the cases of 32 and 64 times faster CPU where a latency of 16 microseconds reduces the performance. In this scenario, the improvement of the processor speed goes up to 16 indicating that it may be important to consider reducing the network requirements if the current serial codes are optimized, for instance, by porting the code to accelerators.

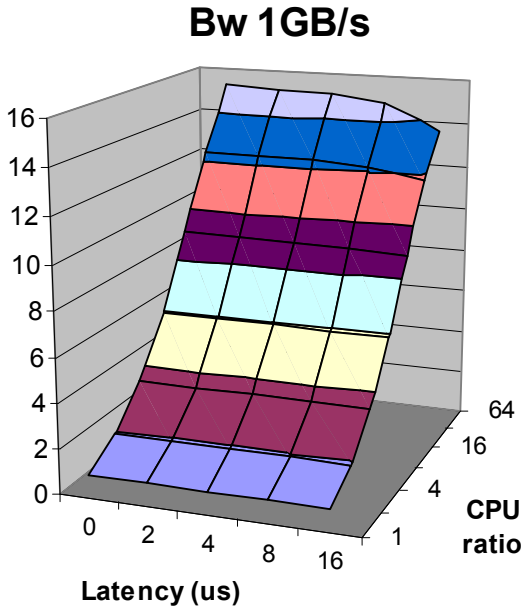


Figure 11: The scenario with a fixed bandwidth of 1GB/s that is quite similar to the current bandwidth on the machine used to obtain the trace file

## 6. Conclusions

The work to refactor the software architecture of DALTON allowed using high-level parallelism that is part of the numerical model in the program. The implementation so far used parallel computations for the calculation of each contribution to the response function. However, the scaling of these single contributions is limited at the moment. The modification of the master-worker design pattern to a team of masters allows calculating several contributions at the same time and increases the scalability of the application accordingly.

For the LSDALTON code we have demonstrated good scalability up to 2048 cores for pure DFT calculations, through the development of MPI integral components, utilizing a newly developed task partitioning scheme, and through the interface to the PBLAS and ScaLAPACK numerical library routines. We have thus established the main code framework and strategies to exploit parallelism at all levels of the code, and thanks to the PRACE community we have learned useful tools for future developments. The results presented clearly show that we are well on our way toward peta-scaling capabilities.

The LSDALTON and DALTON QM codes have also been successfully interfaced to the ChemShell QM/MM package with both binary and directly-linked interfaces. Through ChemShell the DALTON codes have access to a range of MM approaches and supporting functionality to help enable flexible QM/MM modeling on PRACE systems. The LSDALTON interface also supports shared MPI communicators for advanced task-farming parallelization techniques. Benchmark QM/MM calculations using ChemShell/LSDALTON achieve good scaling up to at least 1024 cores.

### Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557 and ScalaLife project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-261523. The work is achieved using the PRACE Research Infrastructure resources Curie and Lindgren.

## References

[1] *DALTON, a molecular electronic structure program, Release DALTON2011 (2011),* http://daltonprogram.org/

[2] LSDALTON, a linear scaling molecular electronic structure program, Release DALTON2011, (2011), http://daltonprogram.org/.

[3]ScalaLife , a life science EU FP7 project www.scalalife.eu

[4] ChemShell, a computational chemistry shell, see www.chemshell.org

[5] *PBLAS, Parallel Basic Linear Algebra Subprograms,* www.netlib.org/pblas.

[6] *ScaLAPACK Scalable Linear Algebra PACKage,* www.netlib.org/scalapack/.

[7] T. W. Keal, P. Sherwood, G. Dutta, A. A. Sokol and C. R. A. Catlow, “Characterization of hydrogen dissociation over aluminium-doped zinc oxide using an efficient massively parallel framework for QM/MM calculations”, Proc. R. Soc. A, 467, 1900 (2011)

[8] J. Kästner, J. M. Carr, T. W. Keal, W. Thiel, A. Wander and P. Sherwood, “DL-FIND: an open-source geometry optimizer for atomistic simulations”, J. Phys, Chem. A, 113, 11856 (2009)

[9] M. F. Guest, I. J. Bush, H. J. J. Van Dam, P. Sherwood, J. M. H. Thomas, J. H. Van Lenthe, R. W. A. Havenith and J. Kendrick, “The GAMESS-UK electronic structure package: algorithms, developments and applications”, Molec. Phys., 103, 719 (2005)

[10] Z. Rinkevicius, O. Vahtras and H. Ågren, "Time-dependent closed and open-shell density functional theory from the perspective of partitioning techniques and projections", J. Mol. Struc. Theochem, 914, 50-59 (2009).

[11] Z. Rinkevicius, K. J. de Almeida, O. Vahtras, "Density functional restricted-unrestricted approach for nonlinear properties: Application to electron paramagnetic resonance parameters of square planar copper complexes", J. Chem. Phys., 129, 064109 (2008).

[12] C. B. Nielsen, O. Christiansen, K. V. Mikkelsen and J. Kongsted, "Density functional self-consistent quantum mechanics/molecular mechanics theory for linear and nonlinear molecular properties: Applications to solvated water and formaldehyde", J. Chem. Phys., 126, 154112 (2007).

[13] O. Vahtras and Z. Rinkevicius, "General excitations in time-dependent density functional theory", J. Chem. Phys., 126, 114101 (2007).

[14] I. Tunell, Z. Rinkevicius, O. Vahtras, P. Salek, T. Helgaker, and Hans Ågren, "Density functional theory of nonlinear triplet response properties with applications to phosphorescence", J. Chem. Phys.,119, 11024, (2003).

[15] P. Salek, S. Høst, L. Thøgersen, P. Jørgensen, P. Manninen, J. Olsen, B. Jansik, S. Reine, F. Pawlowski, E. Tellgren, T. Helgaker, S. Coriani "Linear-scaling implementation of molecular electronic self-consistent field theory" J. Chem. Phys., 126, 11411 (2007)

[16] S. Coriani, S. Høst, B. Jansik, L. Thøgersen, J. Olsen, P. Jørgensen, S. Reine, F. Pawlowski, T. Helgaker, P. Salek "Linear-scaling implementation of molecular electronic self-consistent field theory" J. Chem. Phys., 126, 154108 (2007)

[17] S. Reine, A. Krapp, M. F. Iozzi, V. Bakken, T. Helgaker, F. Pawlowski, P. Swlek, "An efficient density-functional-theory force evaluation for large molecular systems", J. Chem. Phys., 133, 044102 (2010)

[18] J.M. Olsen, K. Aidas, J. Kongsted, "Excited states through polarizable embedding", J. Chem. Theor. Comp., 6, 3721 (2010)

[19] N. A. Murugan, J. Kongsted, Z. Rinkevicius and H. Ågren, "Breakdown of the first hyperpolarizability/bond-length alternation parameter relationship", PNAS, 107, 16453-16458 (2010).

[20] T. Rocha-Rinza, K.Sneskov, O. Christiansen, U. Ryde, J. Kongsted, "Unraveling the similarity of the photoabsorption of deprotonated p-coumaric acid in the gas phase and within the photoactive yellow protein", PhysChemChemPhys, 13, 1585 (2011)

[21] T. Hansen, T. Hansen, K.V. Mikkelsen, J. Kongsted, V. Mujica, "Nonlinear optical effects induced by nano particles in symmetri molecules", J. Phys. Chem. C., 114, 20870 (2010)

[22] X. Aguilar, M. Schliephake, O. Vahtras, J. Gimenez, E.Laure, "Scaling DALTON, a molecular electronic structure program", IEEE 7th International Conference on E-Science, e-Science 2011, Stockholm, Sweden, 256-262 (2011), DOI: 10.1109/eScience.2011.43.